PASSED

Computational Statistics Lab2

Group 6 - Araya Eamrurksiri & Oscar Pettersson

February 10, 2016

Assigment1: Optimizing a model parameter

File mortality_rate.csv contains information about mortaily rates of the fruit flies during a certain period.

1.1 Import `mortality_rate.csv` to R and add one more variable LMR to the data which is the natural logarithm of Rate. Afterwards, divide the data into training and test sets.

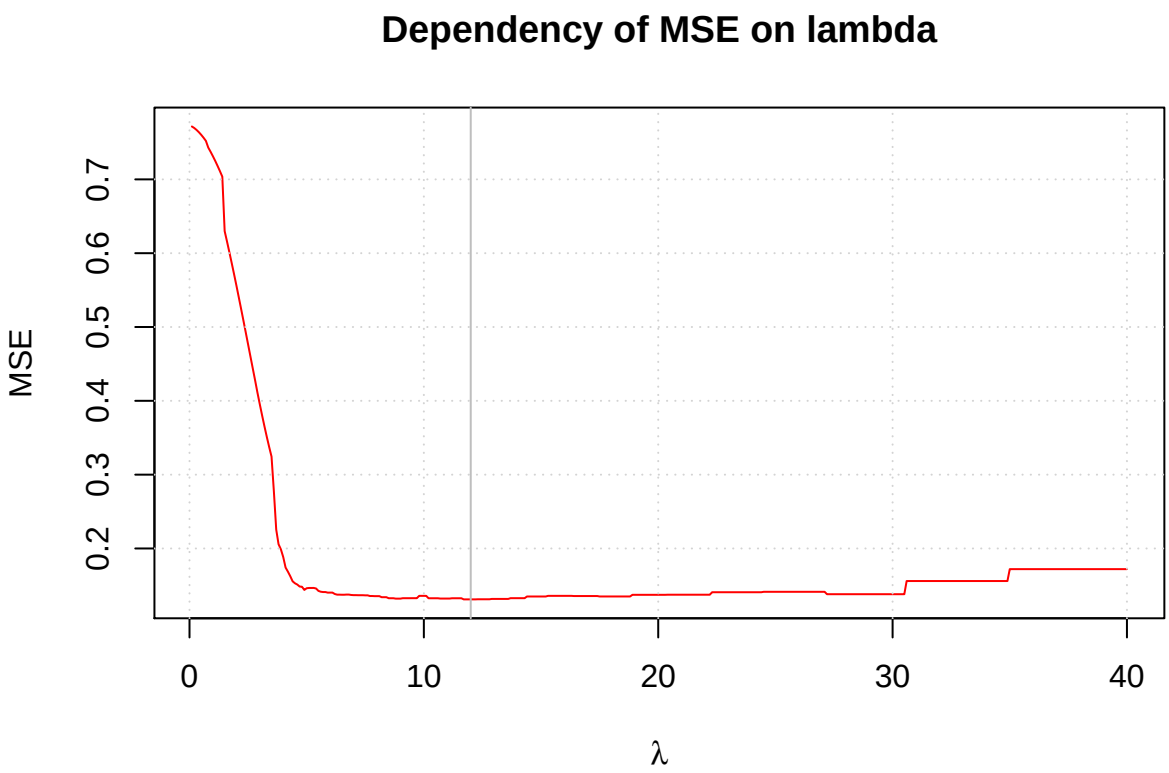
```
mortality$LMR <- log(mortality$Rate)

n=dim(mortality)[1]
set.seed(123456)
id=sample(1:n, floor(n*0.5))
train=mortality[id,]
test=mortality[-id,]
```

1.2 Write your own function myMSE that for given parameters lambda and list pars containing vectors X, Y, Xtest, Ytest fits a LOESS model with response Y and predictor X using loess() function with penalty lambda (parameter enp.target in loess()) and then predicts the model for Xtest. The function should compute the predictive MSE, print it and return as a result.

```
myMSE <- function(pars, lambda){
    fit <- loess(pars[[2]]~pars[[1]], enp.target=lambda)
    pred <- predict(fit, pars[[3]])
    MSE <- sum((pars[[4]]-pred)^2)/length(pars[[3]])
    #print(MSE)
    return(MSE)
}
```

1.3 Use a simple approach: use function myMSE, training and test sets and the following lambda values to estimate the predictive MSE values: Lambda=0.1,0.2,. . . ,40. Create a plot of the MSE values versus lambda and comment on which lambda value is optimal. How many evaluations of myMSE were required to find this value?



The plot above presents the MSE values with different lambda. From looking at this plot, it can be seen that value of lambda around 12 might have the smallest MSE. To be more specific, 0.131047 is the minimum value of MSE. This value is obtained when lambda equals 11.7, 11.8, 11.9, 12, 12.1, 12.2. It is required 117 evaluations in order to find this optimal value but it is required to evaluate all the 400 λ:s in order to know that this is the minimum value.

1.4 Use optimize() function for the same purpose, specify range for search [0.1, 40] and the accuracy 0.01. Have the function managed to find the optimal MSE value? How many myMSE function evaluations were required? Compare to step 3.

```
## $minimum
## [1] 10.69361
##
## $objective
## [1] 0.1321441
```

The function is not being able to find the optimal MSE value but it's quite close. It returns 0.1321441 as the minimum MSE for $\lambda = 10.6936107$. It uses 18 evaluations to find this value. The number of evaluations using the `optimize` function is much smaller than what we have got in step 1.3, only 4.5 %.

1.5 Use optim() function and BFGS method with starting point lambda=35 to find the optimal lambda value. How many myMSE function evaluations were required? Compare the results you obtained with the results from step 4 and make conclusions.

```
## $par
## [1] 35
##
## $value
## [1] 0.1719996
##
## $counts
## function gradient
##        1        1
##
## $convergence
## [1] 0
##
## $message
## NULL
```

Three evaluations are required to find the optimal value but $myMSE$ is only used once for calculating the MSE. The figure in step 1.3 shows that when lambda is equal to 35 the function is already at a local minimum. Thus, there is no need to use so many evaluations to find the local optimal value as it is already got one.

The result we have got from step 1.4 is from starting to find the minimum value of MSE when lambda ranges from 0.1 to 40. Therefore, there have got a better chance to find the optimal value than for the function in this step, where we initially assume that λ is 35.

Assignment2: Maximizing likelihood

File data.RData contains a sample from normal distribution with some parameters μ, σ

2.1 Load the data to R environment.

```
load("data.Rdata")
```

2.2 Write down the log-likelihood function for 100 observations and derive maximum likelihood estimators for μ, σ analytically by setting partial derivatives to zero. Use formulas derived to obtain parameter estimates for the data loaded.

The probability density of the normal distribution is:

$$f(x|\mu,\sigma) = (2\pi\sigma^2)^{-\frac{1}{2}} exp(\frac{-1}{2\sigma^2}(x_j-\mu)^2)$$

The likelihood function is:

$$L(\mu,\sigma;x_1,...,x_n) = (2\pi\sigma^2)^{-\frac{n}{2}} exp(\frac{-1}{2\sigma^2}(\sum_{j=1}^{n}(x_j-\mu)^2))$$

The log-likelihood function is:

$$l(\mu, \sigma; x_1, ..., x_n) = -\frac{n}{2}ln(2\pi) - \frac{n}{2}ln(\sigma^2) - \frac{1}{2\sigma^2}\sum_{j=1}^{n}(x_j - \mu)^2$$

Then, taking the partial derivative with respect to μ, we get

$$\frac{1}{\sigma^2}(\sum_{j=1}^{n} x_j - n\mu)$$

and partial derivative with respect to σ,

$$\frac{1}{\sigma}(\frac{1}{\sigma^2}\sum_{j=1}^{n}(x_j - \mu)^2 - n)$$

The maximum likelihood estimators (setting partial derivative to zero):

$$\hat{\mu} = \frac{\sum_{j=1}^{n} x_j}{n}$$

and

$$\hat{\sigma} = \sqrt{\frac{\sum_{j=1}^{n}(x_j - \hat{\mu})^2}{n}}$$

The parameter estimates, $\hat{\mu}$ and $\hat{\sigma}$, for this data are 1.2755276 and 2.0059765, respectively.

2.3 Now you are assumed to derive maximum likelihood estimates numerically. Why it is a bad idea to maximize likelihood rather than maximizing log-likelihood?

It would be quite difficult to derive maximum likelihood estimates from maximizing the likelihood. Since in order to maximize the function, differentiate the likelihood with respect to a parameter is needed. It is usually more convenient to work with the logarithm of the likelihood rather than likelihood itself as it makes the differentiation much easier. But it is also advisable to use the log-likelihood function for numerical calculations of the optimum points because L is a product of densities for (usually) many observations and they are often close to zero which could lead to the product eventually underflowing to zero for any value of the parameters.

2.4 Optimize the minus log-likelihood function with initial parameters $\mu = 0$, $\sigma = 1$. Try both Conjugate Gradient method and BFGS algorithm with gradient specified and without.

```
#conjugate gradient method with gradient
optim(c(0,1), fn=loglik, gr=gradient, data=data, method="CG")

#BFGS with gradient
optim(c(0,1), fn=loglik, gr=gradient, data=data, method="BFGS")

#conjugate gradient method without gradient
optim(c(0,1), fn=loglik, gr=NULL, data=data, method="CG")

#BFGS without gradient
optim(c(0,1), fn=loglik, gr=NULL, data=data, method="BFGS")
```

2.5 Did algorithms converge in all cases? What were the optimal values of parameters and how many function and gradient evaluations it required for algorithms to converge? Which settings would you recommend?

```
##                      mu    sigma    value function gradient convergence
## CG.gradient    1.275528 2.005976 211.5069       53       17           0
## BFGS.gradient 1.275528 2.005977 211.5069       40       15           0
## CG             1.275528 2.005976 211.5069      210       35           0
## BFGS           1.275528 2.005977 211.5069       37       15           0
```

Both the Conjugate-gradient and BFGS methods, with and without gradient specified, successfully converged. The optimal values of parameters are 1.275528 and 2.005976, respectively, which are the true maximum likelihood estimates. The number of evalutions that are used for function and gradient to converge are shown in the table above. However, when we use the default gradient, our likelihood function is used more times than that in order to calculate the Hessian matrix and to approximate the gradient.

The recommended setting is BFGS method without gradient as it has the least number of evaluations both for the function and the gradient. However, in general it would probably be better to use a known gradient function because *optim*() wouldn't have to calculate Hessians and approximate gradients, so the program should run faster and the result should be more reliable.

Appendix

```
mortality <- read.csv(file = "mortality_rate.csv", header = TRUE, sep = ";", dec = ",")
mortality$LMR <- log(mortality$Rate)

n=dim(mortality)[1]
set.seed(123456)
id=sample(1:n, floor(n*0.5))
train=mortality[id,]
test=mortality[-id,]
myMSE <- function(pars, lambda){
    fit <- loess(pars[[2]]~pars[[1]], enp.target=lambda)
    pred <- predict(fit, pars[[3]])
    MSE <- sum((pars[[4]]-pred)^2)/length(pars[[3]])
    #print(MSE)
    return(MSE)
}
pars <- list(train$Day, train$LMR, test$Day, test$LMR)
lambda <- seq(0.1,40,0.1)
mse <- NULL
for(i in 1:length(lambda)){
    mse[i] <- myMSE(pars, lambda=lambda[i])
}

plot(y=mse, x=lambda, type="l", col="red", pch=20,
     xlab = expression(lambda), ylab = "MSE", main = "Dependency of MSE on lambda")
grid()
abline(v = 12, col="gray")
#lambda[which(mse == min(mse))] #11.7 11.8 11.9 12.0 12.1 12.2 #ans[117:122] = 0.1310470
value <- optimize(f=myMSE, pars=pars, interval=c(0.1,40), tol=0.01)
value
```

```r
value5 <- optim(35, fn=myMSE, pars=pars, method="BFGS")
value5
load("data.Rdata")
n <- length(data)
mu.hat <- sum(data)/n
sigma.hat <- sqrt( (sum((data - mu.hat)^2))/n )
loglik <- function(par, data){
    n <- length(data)
    ll <- -(n*log(2*pi))/2 - (n*log(par[2]^2))/2 - ( sum((data-par[1])^2) )/(2*par[2]^2)
    return(-ll)
}

gradient <- function(par, data){
    n <- length(data)
    mu.hat <- ( sum(data) - (n*par[1]) )/(par[2]^2)
    sigma.hat <- ( ( (1/par[2]^2) * (sum((data - par[1])^2)) ) - n )/( par[2] )
    ans <- c(-mu.hat, -sigma.hat)
    return(ans)
}
## #conjugate gradient method with gradient
## optim(c(0,1), fn=loglik, gr=gradient, data=data, method="CG")
##
## #BFGS with gradient
## optim(c(0,1), fn=loglik, gr=gradient, data=data, method="BFGS")
##
## #conjugate gradient method without gradient
## optim(c(0,1), fn=loglik, gr=NULL, data=data, method="CG")
##
## #BFGS without gradient
## optim(c(0,1), fn=loglik, gr=NULL, data=data, method="BFGS")
#conjugate gradient method with gradient
cg.g <- optim(c(0,1), fn=loglik, gr=gradient, data=data, method="CG")

#BFGS with gradient
bfgs.g <- optim(c(0,1), fn=loglik, gr=gradient, data=data, method="BFGS")

#conjugate gradient method without gradient
cg <- optim(c(0,1), fn=loglik, gr=NULL, data=data, method="CG")

#BFGS without gradient
bfgs <- optim(c(0,1), fn=loglik, gr=NULL, data=data, method="BFGS")

#create table to compare output
CG.gradient <- c(par=cg.g$par, value=cg.g$value, cg.g$counts, convergence=cg.g$convergence)
BFGS.gradient <- c(par=bfgs.g$par, value=bfgs.g$value, bfgs.g$counts, convergence=bfgs.g$convergence)
CG <- c(par=cg$par, value=cg$value, cg$counts, convergence=cg$convergence)
BFGS <- c(par=bfgs$par, value=bfgs$value, bfgs$counts, convergence=bfgs$convergence)

tab <- t(data.frame(CG.gradient,BFGS.gradient,CG,BFGS))
colnames(tab)[1:2] <- c("mu", "sigma")
tab
## NA
```